



# Compiler-based Countermeasure Against Fault Attacks

Thierno Barry, Damien Couroussé, Bruno Robisson

## ► To cite this version:

Thierno Barry, Damien Couroussé, Bruno Robisson. Compiler-based Countermeasure Against Fault Attacks. Workshop on Cryptographic Hardware and Embedded Systems, Sep 2015, Saint-Malo, France. . emse-01232664

**HAL Id: emse-01232664**

**<https://hal-emse.ccsd.cnrs.fr/emse-01232664>**

Submitted on 23 Nov 2015

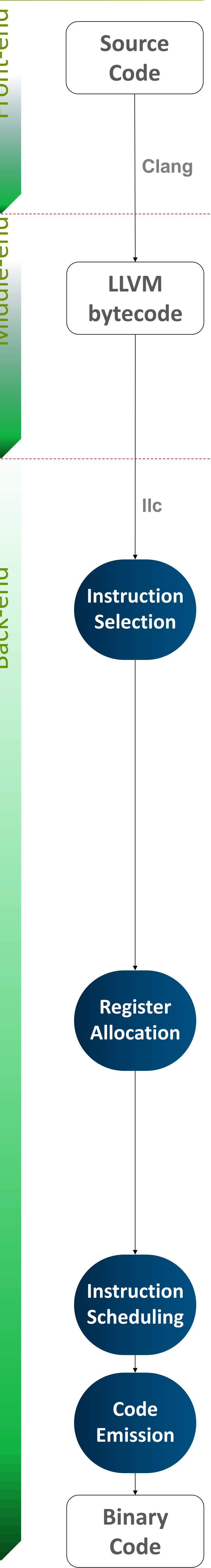
**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CONTEXT

The goal is to implement the instruction duplication technique as a countermeasure against Fault Attacks on an ARM 32-bit Microcontroller[1,2]. Operating inside a compiler allowed us to reduce the security overhead thanks to the flexibility and code transformations opportunities offered by compilers

WORKFLOW



The user identifies the portions of the program to protect

```
@__to_secure__("fault")
int foo(int a, int b){
    . . .
    return a * b + a;
}
```

C source code

The user has a full control over parts of the code to protect

Instructions cannot be duplicated at the middle-end due to the SSA form

```
entry:
    %mul = mul %a, %b
    %add = add %mul, %a
    ret %add
```

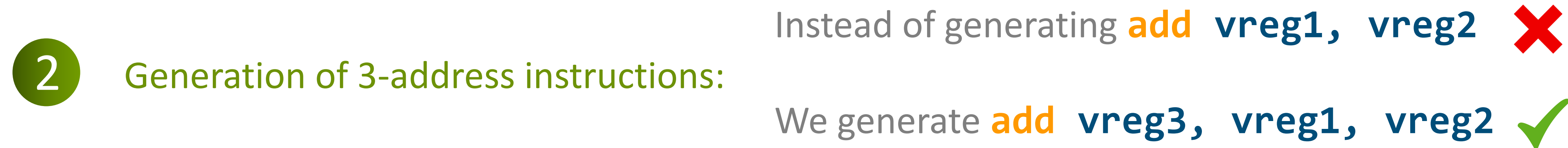
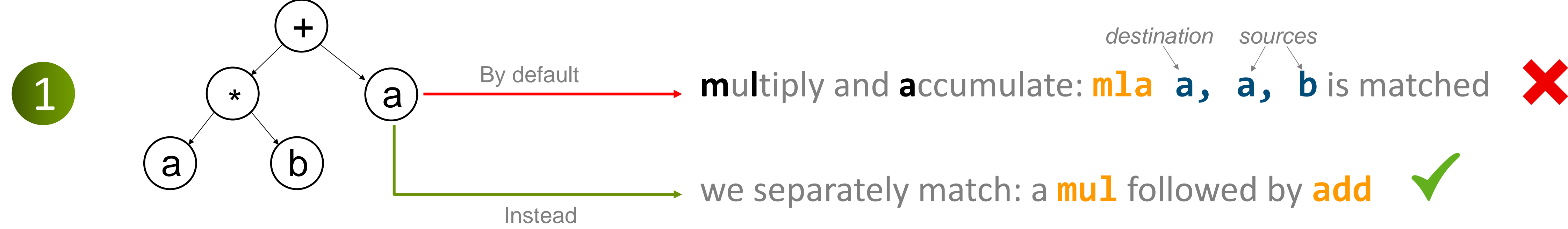
LLVM bytecode

Attempted duplication

```
entry:
    %mul = mul %a, %b
    %mul2 = mul %a, %b
    %add = add %mul, %a
    %add2 = add %mul, %a
```

Unused and will be removed by the Dead Code Elimination pass

We only select instructions that are suitable for duplication



Registers are allocated in favor of duplication

The register allocator tends to reduce *register pressure*: Reusing the allocated registers as soon as possible  
When the liveness intervals (L) of registers are disjoint:  $\{L(\text{vreg3})\} \cap \{L(\text{vreg1}) \cup L(\text{vreg2})\} = \emptyset$

```
add vreg3, vreg1, vreg2 -- By default --> add r0, r0, r1
```

We introduce a constraint:  $\$dst \neq \$src$

```
add r0, r1, r2
```

Instructions are duplicated before scheduling

```
add r0, r1, r2
str r5, [r3, #4]
```

Before duplication

Duplication

```
add r0, r1, r2
add r0, r1, r2
str r5, [r3, #4]
str r5, [r3, #4]
```

Before scheduling

Scheduling

```
str r5, [r3, #4]
add r0, r1, r2
str r5, [r3, #4]
add r0, r1, r2
```

After scheduling

Comparison with assembly approach

	Instruction	Transformation	Duplication
Assembly approach	add r0, r0, r2	mov rx, r0 add r0, rx, r2	mov rx, r0 mov rx, r0 add r0, rx, r2 add r0, rx, r2
Our approach	add r0, r1, r2		add r0, r1, r2 add r0, r1, r2

X 4

X 2

AES 8-bit NIST on ARM Cortex-M3

Unprotected	Protected	Overhead
8541 cycles	17311 cycles	× 2.03

FUTURE WORK & REFERENCES

FUTURE WORK

- Using code annotation for more flexibility when defining the code regions to protect
- Automatic identification of the most vulnerable parts of the program
- compiler-based implementation of the masking countermeasure

REFERENCES

- [1] Barengi et al. *Countermeasures against fault attacks on software implemented AES*  
[2] Moro et al. *Electromagnetic Fault Injection : Towards a Fault Model on a 32-bit Microcontroller*

LEGEND

✓ Duplicable

✗ Not duplicable